

Package: `tablespan` (via `r-universe`)

December 6, 2024

Type Package

Title Create Satisficing 'Excel', 'HTML', 'LaTeX', and 'RTF' Tables
using a Simple Formula

Version 0.1.7

Maintainer Jannik H. Orzek <jannik.orzek@mailbox.org>

Description Create ``good enough" tables with a single formula.
'tablespan' tables can be exported to 'Excel', 'HTML', 'LaTeX',
and 'RTF' by leveraging the packages 'openxlsx' and 'gt'. See
<<https://jhorzek.github.io/tablespan/>> for an introduction.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Imports dplyr, gt, methods, openxlsx, rlang, utils

URL <https://github.com/jhorzek/tablespan>,
<https://jhorzek.github.io/tablespan/>

BugReports <https://github.com/jhorzek/tablespan/issues>

NeedsCompilation no

Author Jannik H. Orzek [aut, cre, cph]
(<<https://orcid.org/0000-0002-3123-2248>>)

Repository CRAN

Date/Publication 2024-12-06 09:20:02 UTC

Config/pak/sysreqs make libicu-dev libxml2-dev libssl-dev libnode-dev

Contents

<code>as_excel</code>	2
<code>as_gt</code>	4

cell_style	5
create_data_styles	6
print.Tablespan	7
tablespan	8
tbl_styles	10

Index	12
--------------	-----------

as_excel	<i>as_excel</i>
----------	-----------------

Description

Write a tablespan table to an excel workbook.

Usage

```
as_excel(
  tbl,
  workbook = openxlsx::createWorkbook(),
  sheet = "Table",
  start_row = 1,
  start_col = 1,
  styles = tbl_styles()
)
```

Arguments

tbl	table created with <code>tablespan::tablespan</code>
workbook	Excel workbook created with <code>openxlsx::createWorkbook()</code>
sheet	name of the sheet to which the table should be written to
start_row	row at which to start the table
start_col	column at which to start the table
styles	openxlsx style for the different table elements (see <code>?tablespan::tbl_styles</code>). The styles element also allows applying custom styles to parts of the data shown in the table body.

Value

openxlsx workbook object that can be edited and saved with openxlsx

Examples

```

library(tablesan)
library(dplyr)
data("iris")

tbl <- tablesan(data = iris[iris$Species == "setosa", ],
  formula = Species ~ (Sepal = Sepal.Length + Sepal.Width) +
    (Petal = (Width = Petal.Length) + Petal.Width))

wb <- as_excel(tbl = tbl)

# saveWorkbook(wb, "iris.xlsx")

# To apply a custom style to some elements use the styles argument. The following
# applies the "bold" style to the rows 1-5 of the Sepal.Length column and
# the rows 9-10 of the Petal.Width column.
bold <- openxlsx::createStyle(textDecoration = "bold")

wb <- as_excel(tbl = tbl,
  styles = tbl_styles(cell_styles = list(cell_style(rows = 1:5,
    colnames = "Sepal.Length",
    style = bold),
    cell_style(rows = 9:10,
    colnames = "Petal.Width",
    style = bold))))

# saveWorkbook(wb, "iris.xlsx")

# The main use case for tablesan is when you already have a summarized table
# that you now want to share using xlsx. The following shows an example using
# the dplyr package:

# First summarize the data:
summarized_table <- mtcars |>
  group_by(cyl, vs) |>
  summarise(N = n(),
    mean_hp = mean(hp),
    sd_hp = sd(hp),
    mean_wt = mean(wt),
    sd_wt = sd(wt))

# Now, we want to create a table, where we show the grouping variables
# as row names and also create spanners for the horse power (hp) and the
# weight (wt) variables:
tbl <- tablesan(data = summarized_table,
  formula = Cylinder:cyl + Engine:vs ~
    N +
    (`Horse Power` = Mean:mean_hp + SD:sd_hp) +
    (`Weight` = Mean:mean_wt + SD:sd_wt),
  title = "Motor Trend Car Road Tests",
  subtitle = "A table created with tablesan",
  footnote = "Data from the infamous mtcars data set.")

```

```

wb <- as_excel(tbl = tbl)

# Create the excel table:
# openxlsx::saveWorkbook(wb,
#                          file = "cars.xlsx", overwrite = TRUE)

```

as_gt

as_gt

Description

Translates a table created with `tablesapn` to a great table (`gt`). See <https://gt.rstudio.com/>.

Usage

```

as_gt(
  tbl,
  groupname_col = NULL,
  separator_style = gt::cell_borders(sides = c("right"), weight = gt::px(1), color =
    "gray"),
  auto_format = TRUE,
  ...
)

```

Arguments

<code>tbl</code>	table created with <code>tablesapn::tablesapn</code>
<code>groupname_col</code>	Provide column names to group data. See <code>?gt::gt</code> for more details.
<code>separator_style</code>	style of the vertical line that separates the row names from the data.
<code>auto_format</code>	should the table be formatted automatically?
<code>...</code>	additional arguments passed to <code>gt::gt()</code> .

Details

`Tablesapn` itself does not provide any printing of tables as HTML table. However, with `as_gt`, `tablesapn` can be translated to a great table which provides html and LaTeX output.

Value

gt table that can be further adapted with the `gt` package.

Examples

```

library(tables)
library(dplyr)
data("mtcars")

summarized_table <- mtcars |>
  group_by(cyl, vs) |>
  summarise(N = n(),
            mean_hp = mean(hp),
            sd_hp = sd(hp),
            mean_wt = mean(wt),
            sd_wt = sd(wt))

tbl <- tables(data = summarized_table,
             formula = (LHS = Cylinder:cyl + Engine:vs) ~
               N +
               (Results = (`Horse Power` = Mean:mean_hp + SD:sd_hp) +
                (`Weight` = Mean:mean_wt + SD:sd_wt)))

gt_tbl <- as_gt(tbl)
gt_tbl

```

cell_style

cell_style

Description

cell_style

Usage

```
cell_style(rows, colnames, style, gridExpand = TRUE, stack = TRUE)
```

Arguments

rows	indices of the rows to which the style should be applied
colnames	names of the columns to which the style should be applied
style	style created with <code>openxlsx::createStyle()</code> that will be applied to the selected cells
gridExpand	see <code>?openxlsx::addStyle</code> : Apply style only to the selected elements (set <code>gridExpand = FALSE</code>) or to all combinations?
stack	should the style be added to existing styles (TRUE) or overwrite existing styles (FALSE)

Value

list with specified styles

Examples

```

library(tables)
data("iris")

tbl <- tables(data = iris[iris$Species == "setosa", ],
             formula = Species ~ (Sepal = Sepal.Length + Sepal.Width) +
             (Petal = (Width = Petal.Length) + Petal.Width))

# To apply a custom style to some elements use the styles argument. The following
# applies the "bold" style to the rows 1-5 of the Sepal.Length column and
# the rows 9-10 of the Petal.Width column.
bold <- openxlsx::createStyle(textDecoration = "bold")

wb <- as_excel(tbl = tbl,
              styles = tbl_styles(cell_styles = list(cell_style(rows = 1:5,
                                                            colnames = "Sepal.Length",
                                                            style = bold),
                                                            cell_style(rows = 9:10,
                                                            colnames = "Petal.Width",
                                                            style = bold))))

# saveWorkbook(wb, "iris.xlsx")

```

create_data_styles *create_data_styles*

Description

This function sets some defaults for `data_styles`. See `?tbl_styles`

Usage

```

create_data_styles(
  double = list(test = is.double, style = openxlsx::createStyle(numFmt = "0.00")),
  integer = list(test = is.integer, style = openxlsx::createStyle(numFmt = "0")),
  ...
)

```

Arguments

<code>double</code>	style for columns of type double
<code>integer</code>	style for columns of type integer
<code>...</code>	add further styles

Details

Styles are applied to the columns in the data set based on their classes (e.g., numeric, character, etc.). `data_styles` must be a list of lists. Each inner list must have two elements: a "test" that is used to determine the class of a data column (e.g., `is.double`) and a style that is then applied to the columns where the test returns TRUE. Note that styles will be applied in the order of the list, meaning that a later style may overwrite an earlier style.

Value

a list of lists with styles

Examples

```
library(tablespace)
# Make all booleans bold:
create_data_styles(boolean = list(test = is.logical,
                                  style = openxlsx::createStyle(textDecoration = "bold")))
```

<code>print.Tablespan</code>	<i>print.Tablespan</i>
------------------------------	------------------------

Description

`print.Tablespan`

Usage

```
## S3 method for class 'Tablespan'
print(x, digits = 2, n = 3, ...)
```

Arguments

<code>x</code>	result from <code>tablespan</code>
<code>digits</code>	number of digits to round doubles to
<code>n</code>	number of rows to print
<code>...</code>	additional arguments passed to <code>prmatrix</code>

Value

nothing

Examples

```
data("iris")
tbl <- tablespan(data = iris[iris$Species == "setosa", ],
                 formula = Species ~ (Sepal = Sepal.Length + Sepal.Width) +
                 (Petal = Petal.Length + Petal.Width))
print(tbl)
```

tablespan	<i>tablespan</i>
-----------	------------------

Description

Create complex table spanners with a simple formula.

Usage

```
tablespan(data, formula, title = NULL, subtitle = NULL, footnote = NULL)
```

Arguments

data	data set
formula	formula to create table
title	string specifying the title of the table
subtitle	string specifying the subtitle of the table
footnote	string specifying the footnote of the table

Details

tablespan provides a formula based approach to adding headers and spanners to an existing data.frame. The objective is to provide a unified, easy to use, but good enough approach to building and exporting tables to Excel, HTML, and LaTeX. To this end, tablespan leverages the awesome packages openxlsx and gt.

Following the tibble approach, tablespan assumes that all items that you may want to use as row names are just columns in your data set (see example). That is, tablespan will allow you to pick some of your items as row names and then just write them in a separate section to the left of the data.

The table headers are defined with a basic formula approach inspired by tables. For example, `Species ~ Sepal.Length + Sepal.Width` defines a table with Species as the row names and Sepal.Length and Sepal.Width as columns. The output will be similar to the following:

```
|Species | Sepal.Length | Sepal.Width|
|:-----|-----:|-----:|
|setosa |          5.1 |          3.5|
|setosa |          4.9 |          3.0|
```

Note that the row names (Species) are in a separate block to the left.

You can add spanner labels with as follows:

```
Species ~ (Sepal = Sepal.Length + Sepal.Width) + (Petal = Sepal.Length + Sepal.Width)
```

This will result in an output similar to:

	Sepal		Petal	
Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	5.1	3.5	1.4	0.2

You can also nest spanners (e.g., `Species ~ (Sepal = (Length = Sepal.Length) + (Width = Sepal.Width))`).

When exporting tables, you may want to rename some of you columns. For example, you may want to rename `Sepal.Length` and `Petal.Length` to `Length` and `Sepal.Width` and `Petal.Width` to `Width`. With `tables`, you can rename the item in the header using `new_name:old_name`. For example, `Species ~ (Sepal = Length:Sepal.Length + Width:Sepal.Width) + (Petal = Length:Sepal.Length + Width:Sepal.Width)` defines a table similar to the following:

	Sepal		Petal	
Species	Length	Width	Length	Width
setosa	5.1	3.5	1.4	0.2

Finally, to create a table without row names, use `1 ~ (Sepal = Length:Sepal.Length + Width:Sepal.Width) + (Petal = Length:Sepal.Length + Width:Sepal.Width)` This defines as table similar to the following:

Sepal		Petal	
Length	Width	Length	Width
5.1	3.5	1.4	0.2

Tables created with `tables` can be exported to Excel (using `openxlsx`), HTML (using `gt`), LaTeX (using `gt`), and RTF (using `gt`).

References:

- `gt`: Iannone R, Cheng J, Schloerke B, Hughes E, Lauer A, Seo J, Brevoort K, Roy O (2024). `gt`: Easily Create Presentation-Ready Display Tables. R package version 0.11.1.9000, <<https://github.com/rstudio/gt>>, <<https://gt.rstudio.com>>.
- `tables`: Murdoch D (2024). `tables`: Formula-Driven Table Generation. R package version 0.9.31, <<https://dmurdoch.github.io/tables/>>.
- `openxlsx`: Schauburger P, Walker A (2023). `_openxlsx: Read, Write and Edit xlsx Files_`. R package version 4.2.5.2, <<https://ycphs.github.io/openxlsx/>>.

Value

Object of class `Tables` with title, subtitle, header info, data, and footnote.

Examples

```
library(tables)
library(dplyr)
data("mtcars")

# We want to report the following table:
```

```

summarized_table <- mtcars |>
  group_by(cyl, vs) |>
  summarise(N = n(),
            mean_hp = mean(hp),
            sd_hp = sd(hp),
            mean_wt = mean(wt),
            sd_wt = sd(wt))

# Create a tablespan:
tbl <- tablespan(data = summarized_table,
                formula = Cylinder:cyl + Engine:vs ~
                  N +
                  (`Horse Power` = Mean:mean_hp + SD:sd_hp) +
                  (`Weight` = Mean:mean_wt + SD:sd_wt),
                title = "Motor Trend Car Road Tests",
                subtitle = "A table created with tablespan",
                footnote = "Data from the infamous mtcars data set.")

tbl

# Export as Excel table:
wb <- as_excel(tbl = tbl)

# Save using openxlsx
# openxlsx::saveWorkbook(wb, "iris.xlsx")

# Export as gt:
gt_tbl <- as_gt(tbl = tbl)
gt_tbl

```

tbl_styles

*tbl_styles***Description**

Define styles for different elements of the table.

Usage

```
tbl_styles(
  background_style = openxlsx::createStyle(fgFill = "#ffffff"),
  hline_style = openxlsx::createStyle(border = "Top", borderColour =
    openxlsx::openxlsx_getOp("borderColour", "black"), borderStyle =
    openxlsx::openxlsx_getOp("borderStyle", "double")),
  vline_style = openxlsx::createStyle(border = "Left", borderColour =
    openxlsx::openxlsx_getOp("borderColour", "black"), borderStyle =
    openxlsx::openxlsx_getOp("borderStyle", "double")),
  title_style = openxlsx::createStyle(fontSize = 14, halign = "left", textDecoration =
    "bold"),

```

```

    subtitle_style = openxlsx::createStyle(fontSize = 11, halign = "left", textDecoration =
      "bold"),
    header_style = openxlsx::createStyle(fontSize = 11, halign = "center", border =
      "BottomLeftRight", borderColour = openxlsx::openxlsx_getOp("borderColour", "black"),
      borderStyle = openxlsx::openxlsx_getOp("borderStyle", "double"), textDecoration =
      "bold"),
    merge_rownames = TRUE,
    merged_rownames_style = createStyle(valign = "top"),
    footnote_style = openxlsx::createStyle(fontSize = 11, halign = "left"),
    data_styles = create_data_styles(),
    cell_styles = NULL
  )

```

Arguments

background_style color etc. for the entire background of the table

hline_style style for the horizontal lines used in the table. Note: the style for the lines under spanners is defined in the title_style.

vline_style style for the vertical lines used in the table. Note: the style for the lines under spanners is defined in the title_style.

title_style style applied to the table title

subtitle_style style applied to the table subtitle

header_style style applied to the table header (column names)

merge_rownames boolean: Should adjacent rows with identical names be merged?

merged_rownames_style style applied to the merged rownames

footnote_style style applied to the table footnote

data_styles styles applied to the columns in the data set based on their classes (e.g., numeric, character, etc.). data_styles must be a list of lists. Each inner list must have two elements: a "test" that is used to determine the class of a data column (e.g., is.double) and a style that is then applied to the columns where the test returns TRUE. Note that styles will be applied in the order of the list, meaning that a later style may overwrite an earlier style.

cell_styles an optional list with styles for selected cells in the data frame.

Value

a list with styles for different elements of the table

Examples

```
tbl_styles()
```

Index

`as_excel`, 2

`as_gt`, 4

`cell_style`, 5

`create_data_styles`, 6

`print.Tablespan`, 7

`tablespan`, 8

`tbl_styles`, 10